МГТУ им. Н. Э. Баумана, кафедра ИУ5
курс “Разработка интернет-приложений”

Лабораторная работа №3
Python. Функциональные возможности

Москва 2018

Задание и порядок выполнения

В этой лабораторной работе вы познакомитесь с популярной СУБД MySQL, создадите свою базу данных. Также вам нужно будет дополнить свои классы предметной области, связав их с созданной базой. После этого вы создадите свои модели с помощью Django ORM, отобразите объекты из БД с помощью этих моделей и ClassBasedViews.

Для сдачи вы должны иметь:

1. Скрипт с подключением к БД и несколькими запросами.
2. Набор классов вашей предметной области с привязкой к СУБД (класс должен уметь хотя бы получать нужные записи из БД и преобразовывать их в объекты этого класса)
3. Модели вашей предметной области
4. View для отображения списка ваших сущностей

Теория и примеры

Установка MySQL и простые операции

MySQL - одна из самых популярных классических реляционных СУБД, используемых в современном вебе.

Синтаксис и набор операторов в MySQL можно найти на сайте http://dev.mysql.com/doc/ в разделе документации. Последней является версия MySQL 5.7, однако для работы обычно рекомендуется использовать предыдущую, стабильную версию, например 5.6 или 5.5.

Для работы вам потребуется установить MySQL на компьютер. Если у вас Windows, потребуется скачать дистрибутив с сайта http://dev.mysql.com/downloads/windows/. Также вам потребуется коннектор для Python, чтобы выполнять запросы из кода. В большинстве случаев для проектирования баз данных бывает полезна программа MySQL Workbench, где можно визуализировать схему базы в виде ER-диаграм. Нечто подобное вы могли видеть в MS Access.

После того, как вы установили MySQL, необходимо запустить сервер. В случае с Windows можно воспользоваться этим мануалом: http://dev.mysql.com/doc/refman/5.7/en/windows-start-command-line.html

Теперь можно работать с MySQL! Если вы установили Workbench, можно открыть его, подсоединиться к серверу и выполнять запросы из интерфейса. В данном методическом материале все примеры будут рассматриваться в командной строке.

Чтобы начать работать с сервером MySQL, нужно подсоединиться к нему из клиента. Для этого можно открыть Workbench или использовать командную строку ("C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql" в случае с windows)

Некоторые стандартные необходимые команды:

show databases - показать доступные базы данных
use <имя базы> - выбрать базу данных
show tables - показать таблицы в выбранной базе данных
describe <имя таблицы> - вывести схему таблицы

Для начала необходимо создать пользователя базы данных. У него будет доступ к этой БД.

Туториал о том, как это делается: https://www.digitalocean.com/community/tutorials/mysql-ru

CREATE USER ‘dbuser’@'localhost' IDENTIFIED BY '123';

После этого нужно создать базу данных.

CREATE DATABASE `first_db` CHARACTER SET utf8 COLLATE utf8_general_ci;

Список операторов и опций можно найти здесь: http://www.spravkaweb.ru/mysql/sql/createdb

И выдать права новому пользователю на эту базу данных.

GRANT ALL PRIVILEGES ON first_db.* TO 'dbuser'@'localhost';

Теперь можно создавать таблицу в этой базе данных, но сначала нужно в нее перейти:

```
mysql> use first_db;
Database changed
```

Чтобы создать таблицу нужно использовать команду create table. Рассмотрим простой пример с созданием таблицы “книга”, в которой будет название книги и ее описание:

Мануал по оператору create table: http://www.spravkaweb.ru/mysql/sql/createtable

```
mysql> CREATE TABLE
    ->     `books` (
    ->         `id` INT(11) NOT NULL AUTO_INCREMENT,
    ->         `name` CHAR(30) NOT NULL,
    ->         `description` CHAR(255) NOT NULL,
    ->         PRIMARY KEY(`id`)
    ->     )
    ->
    -> ;
Query OK, 0 rows affected (0,22 sec)

mysql> show tables;
+--------------------+
| Tables_in_first_db |
+--------------------+
| books              |
+--------------------+
1 row in set (0,00 sec)
```

Для добавления/чтения/обновления/удаления (CRUD) записей служат команды INSERT, DELETE, UPDATE, SELECT. Подробно эти команды

рассмотрены в курсе баз данных.

Краткое описание команд можно найти здесь: http://www.mysql.ru/docs/man/Data_Manipulation.html

Добавим запись и получим ее из БД:

```
mysql> INSERT INTO books VALUES(1, 'Война и Мир', 'Книга Толстого');
Query OK, 1 row affected (0,05 sec)

mysql> SELECT * FROM books;
+----+---------------------+----------------------------+
| id | name                | description                |
+----+---------------------+----------------------------+
|  1 | Война и Мир         | Книга Толстого             |
+----+---------------------+----------------------------+
1 row in set (0,01 sec)
```

Остальные команды и различные вариации операторов остаются на самостоятельное изучение.

Обращение к БД из Python.

В этой части лабораторной работы необходимо создать подключение из Python к MySQL, занести и выбрать несколько записей с помощью кода.

Для начала требуется установить пакет mysqlclient из pip. Это необходимый набор классов и функций для работы с mysql из вашего кода.

Ссылка на репозиторий на github: https://github.com/PyMySQL/mysqlclient-python

Команда для установки: pip install mysqlclient

В этой части вашей задачей является написание простого скрипта, который подключается к базе данных, добавляет одну запись, затем получает и выводит на экран все записи таблицы books, а затем удаляет все записи.

Документация по mysqlclient: https://mysqlclient.readthedocs.io/en/latest/user_guide.html

В качестве примера напишем простой код:

```python
import MySQLdb

# Открываем соединение
db = MySQLdb.connect(
    host="localhost",
    user="dbuser",
    passwd="123",
    db="first_db"
)

# Получаем курсор для работы с БД
c = db.cursor()

# Выполняем вставку
c.execute("INSERT INTO books (name, description) VALUES (%s, %s);", ('Книга', 'Описание книги'))
# Фиксируем изменения
db.commit()

# Выполняем выборку
c.execute("SELECT * FROM books;")

# Забираем все полученные записи
entries = c.fetchall()

# И печатаем их
for e in entries:
    print(e)

c.close()   # Закрываем курсор
db.close()  # Закрываем соединение
```

Результат работы:

(1, 'Война и Мир', 'Книга Толстого')
(3, 'Книга', 'Описание книги')

Написание классов предметной области с соединением с БД

В этой части вам нужно добавить возможность выборки из базы объектов вашей предметной области. В предыдущих лабораторных работах вы делали классы вашей предметной области. Теперь нужно добавить методы для получения этих объектов из БД.

Для удобства и переиспользования кода завернем код с соединением к БД из прошлого примера в новый класс:

```
class Connection:
    def __init__(self, user, password, db, host='localhost'):
        # Сохраняем параметры соединения
        self.user = user
        self.host = host
        self.password = password
        self.db = db
        self._connection = None

    @property
    def connection(self):
        return self._connection

    def __enter__(self):
        self.connect()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.disconnect()

    def connect(self):
        """
        Открытие соединения
        """
        if not self._connection:
            self._connection = MySQLdb.connect(
                host=self.host,
                user=self.user,
                passwd=self.password,
                db=self.db
            )

    def disconnect(self):
        """
        Закрытие соединения
        """
        if self._connection:
            self._connection.close()
```

Рассмотрим простой код, в котором создадим класс для книги и сделаем метод сохранения книги в БД.

```
class Book:

    def __init__(self, db_connection, name, description):
        # Сохраняем соединение и данные книги
        self.db_connection = db_connection.connection
        self.name = name
        self.description = description

    def save(self):
        # Записываем данные из объекта книги в запись БД.
        c = self.db_connection.cursor()
        c.execute("INSERT INTO books (name, description) VALUES (%s, %s);",
                  (self.name, self.description))
        self.db_connection.commit()
        c.close()


con = Connection("dbuser", "123", "first_db")

with con:
    book = Book(con, 'Новая книга', 'Описание новой книги')
    book.save()
```

В результате работы скрипта получаем новую запись в MySQL:

```
mysql> SELECT * FROM books;
+----+----------------------+----------------------------------------+
| id | name                 | description                            |
+----+----------------------+----------------------------------------+
|  1 | Война и Мир          | Книга Толстого                         |
|  5 | Новая книга          | Описание новой книги                   |
```

Django ORM

Django предоставляет удобные возможности для представления базы данных в виде python-объектов. Вся работа, которая описана в предыдущих примерах, уже реализована в django-моделях! Не нужно писать свои классы-обертки для записей БД, не нужно писать коннекторы. Django ORM облегчает и ускоряет разработку. Однако необходимо помнить, что в случае реализации сложных SQL-запросов бывает быстрее и выгоднее использовать чистые запросы без ORM.

Для использования ORM требуется описать свои модели предметной области в виде классов, наследованных от django.db.models.Model.

Примеры работы с ORM:
https://docs.djangoproject.com/es/1.10/topics/db/models/

Опишем простую модель для книги:

```
class BookModel(models.Model):
    name = models.CharField(max_length=30)
    description = models.CharField(max_length=255)
```

Чтобы методы модели работали, необходимо указать настройки подключения БД в settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': <имя бд>,
        'USER': <имя пользователя>,
        'PASSWORD': <пароль пользователя>,
        'HOST': <хост, где расположена СУБД>,
        'PORT': 3306, # Стандартный порт MySQL
        'OPTIONS': {'charset': 'utf8'},
        'TEST_CHARSET': 'utf8',
    }
}
```

В этой части лабораторной работы требуется самостоятельно создать модели по предметной области своего домашнего задания.

Все запросы к БД представляются в Django ORM в виде объектов QuerySet. Это своеобразный “конструктор” запросов, который позволяет с помощью кода “собрать” SQL-запрос. Примеры работы с queryset и моделями можно найти здесь: https://docs.djangoproject.com/el/1.10/ref/models/querysets/

Кроме возможности создать модели и управлять ими из кода, Django ORM также позволяет создавать БД по описанию моделей, а также изменять структуру БД при изменении моделей.

Для этих действий используются так называемые миграции. Это скрипты, которыу выполняют преобразование схемы базы данных с помощью ALTER TABLE.

Миграции в Django создаются с помощью команды manage.py makemigrations <название приложения>. После того, как миграция создана (скрипт миграции создался и добавился в папку migrations), ее нужно применить с помощью команды manage.py migrate <название приложения>.

Все изменения моделей (или их создание) будут фиксироваться в миграции. Если модели до миграции не было, значит после применения миграции будет создана соответствующая таблица. Если модель изменена (например, добавлено поле), после применения миграции это поле будет добавлено в соответствующую таблицу.

Class Based Views

До этого момента все view, которые вы писали, представляли собой простые функции, которые принимают запрос и рендерят ответ.

В Django существует механизм view, основанных на классах, которые уже заточены на конкретное поведение. Например, чтобы отобразить список объектов, есть ListView. Для отображения формы изменения объекта есть EditView.

Документацию можно найти здесь: https://docs.djangoproject.com/en/1.10/topics/class-based-views/

Список всех встроенных ClassBasedViews: https://docs.djangoproject.com/ja/1.10/ref/class-based-views/

В этой части задания вам необходимо создать view, которая будет отображать список объектов вашей предметной области на отдельной страничке.

Выполнение ЛР

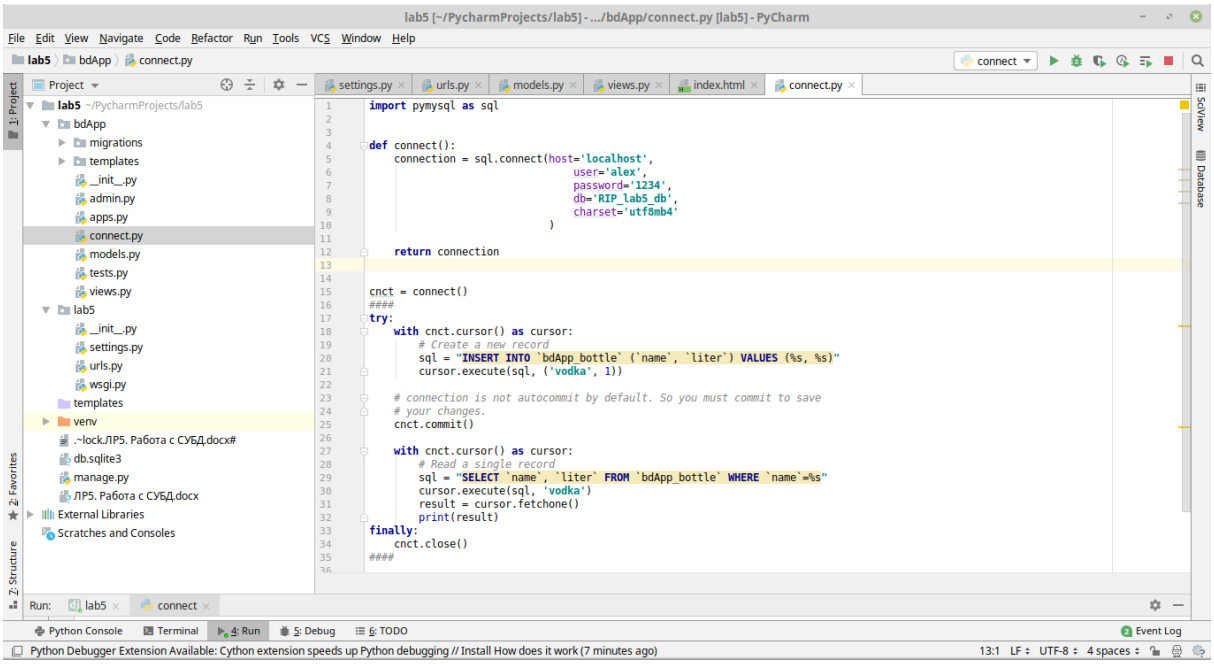
```
from django.contrib import admin
from django.urls import path
from bdApp import views as V
urlpatterns = [
    path('admin/', admin.site.urls),
    path('look_base/', V.BottleView.as_view())
]
```

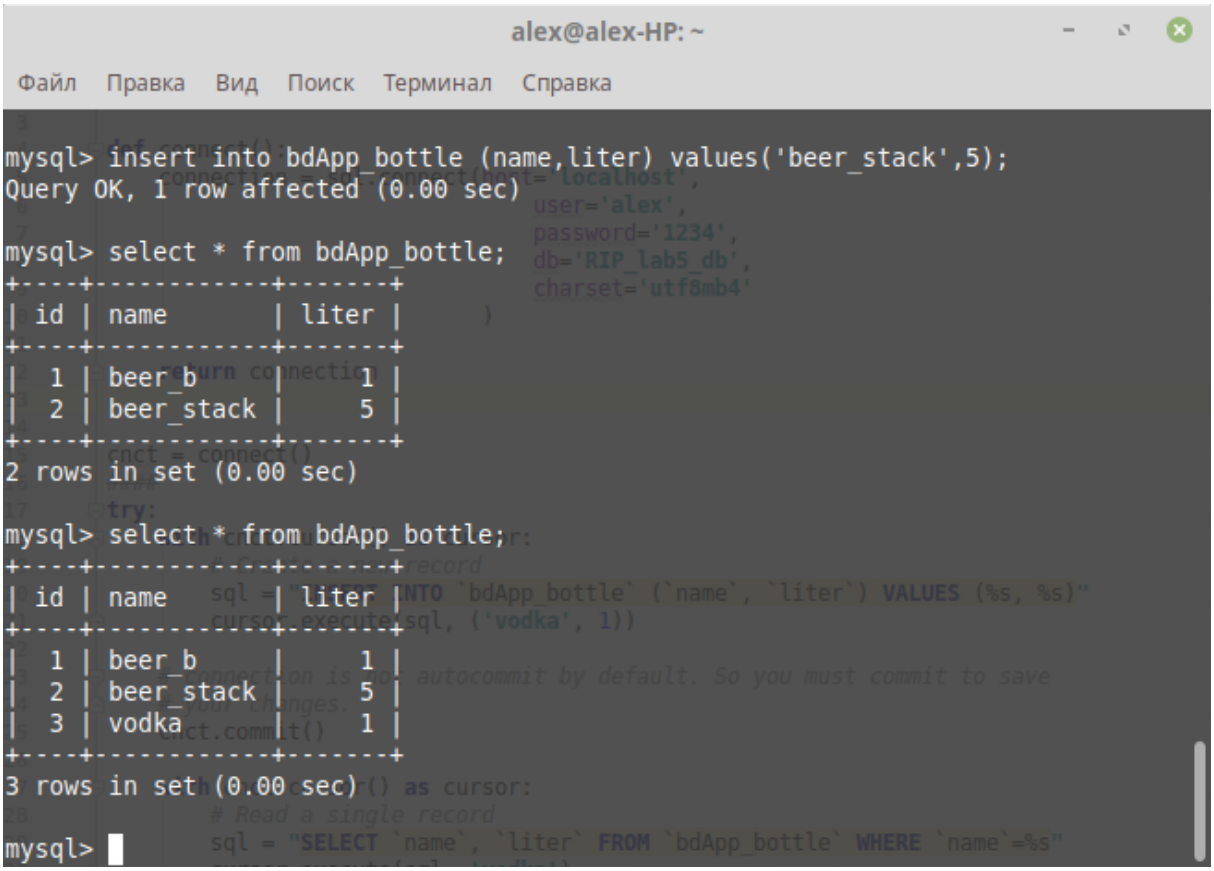
```
from django.db import models
class Bottle(models.Model):
    id = models.AutoField(primary_key=True,unique=True)
    name = models.CharField(max_length=50)
    liter = models.IntegerField()
```

```
from django.shortcuts import render, HttpResponse
from django.views import View
from bdApp.models import Bottle
# Create your views here.
class BottleView(View):
    def get(self,request):
        objects = Bottle.objects.all()
        return render(request,'index.html',{'obj': objects})
```

```
import pymysql as sql
def connect():
    connection = sql.connect(host='localhost',
                                user='alex',
                                password='1234',
                                db='RIP_lab5_db',
                                charset='utf8mb4'
                            )
    return connection
cnct = connect()
####
try:
    with cnct.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO `bdApp_bottle` (`name`, `liter`) VALUES (%s,
%s)"
```

```
            cursor.execute(sql, ('vodka', 1))
        # connection is not autocommit by default. So you must commit to save
        # your changes.
        cnct.commit()
        with cnct.cursor() as cursor:
            # Read a single record
            sql = "SELECT `name`, `liter` FROM `bdApp_bottle` WHERE `name`=%s"
            cursor.execute(sql, 'vodka')
            result = cursor.fetchone()
            print(result)
finally:
    cnct.close()
```




alex@alex-HP: ~
Файл Правка Вид Поиск Терминал Справка
mysql> insert into bdApp_bottle (name,liter) values('beer_stack',5);
Query OK, 1 row affected (0.00 sec)
mysql> select * from bdApp_bottle;
+----+------------+-------+
| id | name | liter |
+----+------------+-------+
| 1 | beer_b | 1 |
| 2 | beer_stack | 5 |
+----+------------+-------+
2 rows in set (0.00 sec)
mysql> select * from bdApp_bottle;
+----+------------+-------+
| id | name | liter |
+----+------------+-------+
1	beer_b	1
2	beer_stack	5
3	vodka	1
+----+------------+-------+
3 rows in set (0.00 sec)
mysql>